

# Schedulability Analysis of Periodic Fixed Priority Systems

Enrico Bini, Giorgio C. Buttazzo

**Abstract**—Feasibility analysis of fixed priority systems has been widely studied in the real-time literature and several acceptance tests have been proposed to guarantee a set of periodic tasks. They can be divided in two main classes: polynomial time tests and exact tests. Polynomial time tests can efficiently be used for on-line guarantee of real-time applications, where tasks are activated at runtime. These tests introduce a negligible overhead, when executed upon a new task arrival, however provide only a sufficient schedulability condition, which may cause a poor processor utilization. On the other hand, exact tests, which are based on response time analysis, provide a necessary and sufficient schedulability condition, but are too complex to be executed on line for large task sets. As a consequence, for large task sets, they are often executed off line.

This paper proposes a novel approach for analyzing the schedulability of periodic task sets on a single processor under an arbitrary fixed priority assignment. Using this approach, we derive a new schedulability test which can be tuned through a parameter to balance complexity vs. acceptance ratio, so that it can be used on line to better exploit the processor, based on the available computational power. Extensive simulations show that our test, when used in its exact form, is significantly faster than the current response time analysis methods.

Moreover the proposed approach, for its elegance and compactness, offers an explanation of some known phenomena of fixed priority scheduling and could be helpful for further work on schedulability analysis.

**Index Terms**—D.4.7.e Real-time systems and embedded, K.6.1.e Systems analysis and design, Fixed priority scheduling.

## I. INTRODUCTION

Fixed priority scheduling is widely used in modern real-time systems, since it can be easily implemented on top of commercial kernels that provide a limited number of priority levels. One of the most common fixed priority assignment follows the Rate Monotonic (RM) algorithm, according to which tasks' priorities are ordered based on tasks' activation rates, so that the task with the shortest period is assigned the highest priority. In general, however, there can be particular situations in which task priorities, although fixed, do not necessarily follow the Rate Monotonic assignment.

Liu and Layland [1] proved that in a uniprocessor system RM is optimal among all fixed priority schemes, meaning that if a task set is not schedulable by RM, then it cannot be scheduled by any other fixed priority assignment. In the same paper, the authors also derived a simple guarantee test to verify the feasibility of a set of  $n$  periodic tasks under RM. Each periodic task  $\tau_i$  consists of an infinite sequence of jobs  $\tau_{ik}$  ( $k = 1, 2, \dots$ ), where the first job  $\tau_{i1}$  is released at time  $r_{i1} = \Phi_i$  (the task phase) and the generic  $k^{\text{th}}$  job  $\tau_{ik}$  is released at time  $r_{ik} = \Phi_i + (k - 1)T_i$ , where  $T_i$  is

the task period. Each job is characterized by a worst-case execution time  $C_i$ , a relative deadline  $D_i$  and an absolute deadline  $d_{ik} = r_{ik} + D_i$ . The ratio  $U_i = C_i/T_i$  is called the *utilization factor* of task  $\tau_i$  and represents the fraction of processor time used by that task. Finally, the value

$$U_p = \sum_{i=1}^n U_i$$

is called the *total processor utilization factor* and represents the fraction of processor time used by the periodic task set. Clearly, if  $U_p > 1$  no feasible schedule exists for the task set.

The schedulability condition for RM is derived for a set  $\Gamma_n$  of  $n$  periodic tasks under the assumptions that all tasks start simultaneously at time  $t = 0$ , relative deadlines are equal to periods and tasks are independent, meaning that they do not have resource constraints, nor precedence relations. Under such assumptions, a set of  $n$  periodic tasks is schedulable by the RM algorithm if

$$\sum_{i=1}^n U_i \leq n(2^{1/n} - 1). \quad (1)$$

Throughout the paper, we will refer to the previous schedulability condition as the LL-test. We recall that

$$\lim_{n \rightarrow \infty} n(2^{1/n} - 1) = \ln 2 \simeq 0.69.$$

After this first result, a lot of work has been done to improve the schedulability bound of the RM algorithm or relax some restrictive assumption on the task set.

Lehoczy, Sha, and Ding [2] performed a statistical study and showed that for task sets with randomly generated parameters the LL-test is able to accept schedulable task sets with an average breakdown utilization of about 88%. Exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived [3], [2], [4], [5]. Using the method proposed by Audsley et al. [4], a periodic task set is schedulable with the RM algorithm if and only if the worst-case response time of each task is less than or equal to its deadline. The worst-case response time  $R_i$  of a task can be computed using the following iterative formula:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = C_i + \sum_{j:D_j < D_i} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j \end{cases} \quad (2)$$

where the worst-case response time of task  $\tau_i$  is given by the smallest value of  $R_i^{(k)}$  such that  $R_i^{(k)} = R_i^{(k-1)}$ . It is worth noting, however, that the complexity of the exact

test is pseudo-polynomial, thus it may be unsuited for on-line admission control, especially in those real-time applications consisting of large task sets running on slow micro-processors. Leung and Whitehead [6] considered the case of deadlines smaller than periods and proved that the Deadline Monotonic priority assignment is optimal. Sha, Rajkumar and Lehozcky [7] extended the Rate Monotonic analysis in the presence of resource constraints, where access to resources is performed using concurrency control protocols, such as the Priority Inheritance Protocol and the Priority Ceiling Protocol. Audsley et al. [4] generalized the response time analysis including resource constraints. Burns, Davis, and Punnekats [8] extended it to take fault-tolerant constraints into account. Sjödin and Hansson [9] provided some methods for reducing the number of iterations in computing the tasks response times, however the worst-case complexity of their test is still pseudo-polynomial.

In the last years, other authors [10], [11], [12] proposed novel approaches for deriving polynomial time tests with better acceptance ratio than the LL-test. For example, the Hyperbolic Bound (HB) proposed by Bini et al. [12] improves the acceptance ratio by a factor of  $\sqrt{2}$  for large  $n$ , compared with the Liu and Layland test. According to HB method, a set of periodic tasks is schedulable by RM if

$$\prod_{i=1}^n (U_i + 1) \leq 2. \quad (3)$$

The authors also extended the test in the presence of resource constraints and aperiodic servers.

The Liu and Layland utilization bound has been improved by considering some additional information on the task set, such as the number of harmonic chains and the value of the periods [13], [14]. A similar approach was proposed by Park et al. [15] and, in the case of a graph structured task, by Liu and Hu [16].

In this paper, we propose a novel and more general approach for analyzing the schedulability of periodic task sets under the Rate Monotonic priority assignment. Using this approach, we derived a new schedulability test, called  $\delta$ -HET (Hyperplanes  $\delta$ -Exact Test), which can be tuned through a parameter to balance complexity vs. acceptance ratio, so that it can be used on-line to better exploit the processor, based on the available computational power. So, for example, if the processor is powerful enough, and there is sufficient time for on-line guarantee, our test can be tuned to behave like the response time test, but with less execution overhead. On the other hand, when the processor utilization is high and the overhead of the guarantee test must be contained, our test can be set to run in less time, with decreased performance.

Extensive simulations show that our test, when used in its exact form, is significantly faster than the current response time analysis methods, and performs much better than polynomial tests (such LL or HB) when used in its reduced form.

As a last remark, we like to notice that the main result of this paper, expressed by Theorem 3, is very general and provides a new view of the some results present in the literature [13], [15], [14].

The rest of the paper is organized as follows. Section II describes the task model and states our notation. Section III explains our approach in detail. Section IV illustrates the proof of the main result of the paper. Section V introduces the tunable guarantee test, called the  $\delta$ -HET test. Section VI compares the  $\delta$ -HET test with others similar tests proposed in the literature. Section VII disusses some extensions that allow to apply the proposed approach to more realistic cases. Finally, Section VIII presents our conclusions and future work.

## II. TASK MODEL AND NOTATION

In this section we introduce the task model and the notation we will use throughout the paper. We consider a set  $\Gamma_n = \{\tau_1, \dots, \tau_n\}$  of  $n$  periodic (or sporadic) tasks, where each task  $\tau_i = (\Phi_i, C_i, T_i, D_i)$  is characterized by an initial activation time  $\Phi_i$  (phase), a worst-case computation time  $C_i$ , a period (or a minimum interarrival time)  $T_i$ , and a relative deadline  $D_i$  not greater than  $T_i$ . We denote a fixed-priority assignment as FP and, without loss of generality, we assume the tasks in  $\Gamma_n$  are ordered by decreasing priority, so that  $\tau_1$  is the highest priority task. If priorities are assigned based on the RM algorithm, then we have FP=RM. Each task  $\tau_i$  consists of an infinite sequence of jobs, where  $\tau_{ik}$  denotes the  $k^{\text{th}}$  job of task  $\tau_i$ . In particular,  $r_{ik}$  and  $f_{ik}$  denote the release time and the finishing time of  $\tau_{ik}$ , respectively. Each job has a relative deadline  $D_i \leq T_i$ , thus, for the periodicity assumption, the release time  $r_{ik}$  and the absolute deadline  $d_{ik}$  of job  $\tau_{ik}$  can be computed as follows:

$$r_{ik} = \Phi_i + (k - 1)T_i, \quad d_{ik} = r_{ik} + D_i.$$

Finally,  $U_i = C_i/T_i$  denotes the utilization factor of task  $\tau_i$ .

As proved by Liu and Layland [1], the worst-case scenario for a periodic task set scheduled by RM occurs when all the tasks are simultaneously activated at the same time. Hence, without loss of generality, we assume  $\Phi_i = 0$  for all the tasks and then we will denote the task  $\tau_i = (0, C_i, T_i, D_i)$  simply by  $(C_i, T_i, D_i)$ . It is not difficult to see that under arbitrary fixed priorities (non-RM) this scenario of simultaneous activation is still the worst. Moreover, it has been proved that such a condition represents the worst-case scenario also in the presence of shared resources [7].

To simplify the presentation of the proposed approach, we initially assume that tasks cannot be blocked and are fully preemptive. The presence of blocking times and the extension to non-preemptive scheduling will be considered in Section VII.

In our formulation, a task set is viewed as a point in a specific space of the task set parameters, hence the feasibility test will be expressed as a check that verifies whether a point belongs to a region  $\mathbb{M}_n$  of the FP schedulable tasks sets. In particular, region  $\mathbb{M}_n$  is defined as follows:

$$\mathbb{M}_n(T_1, \dots, T_n, D_1, \dots, D_n) = \{(C_1, \dots, C_n) \in \mathbf{R}_+^n : \Gamma_n \text{ is schedulable by FP}\}. \quad (4)$$

We note that periods  $T_i$  and deadlines  $D_i$  are considered as parameters, whereas  $C_i$  are the free variables. Hence, we obtain a constraint on the  $C_i$  variables, which is a function of

all the  $T_i$  and the  $D_i$ . The space in which every coordinate is represented by a task computation time  $C_i$ , is called the  $C$ -space. In the following section, we express region  $\mathbb{M}_n$  in a convenient form which simplifies the feasibility check.

### III. EXPRESSING $\mathbb{M}_n$

A first attempt to analytically characterize the  $\mathbb{M}_n$  region in the  $C$ -space was indirectly done by Lehoczky, Sha and Ding [2]. In their work, deadlines are assumed equal to periods, whereas in this paper, from Section IV on, we consider the more general case of  $D_i \leq T_i$ . They proved the following theorem [2]:

*Theorem 1 (Theorem 2 in [2]):* Given a periodic task set  $\Gamma_n = \{\tau_1, \dots, \tau_n\}$ ,

- 1)  $\tau_i$  is feasibly schedulable (for any task phasing) by the RM algorithm if and only if:

$$L_i = \min_{t \in \mathcal{S}_i} \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \leq 1$$

where  $\mathcal{S}_i = \{rT_j : j = 1 \dots i, r = 1 \dots \lfloor \frac{T_i}{T_j} \rfloor\}$ .

- 2) The entire task set is feasibly schedulable (for any task phasing) by RM if and only if:

$$\max_{i=1 \dots n} L_i \leq 1.$$

Manipulating this result, we can restate the theorem in a more expressive form (in the next mathematical passages we will widely use the logical OR operator  $\vee$  and the logical AND operator  $\wedge$ ):

$$\begin{aligned} & \max_{i=1 \dots n} \min_{t \in \mathcal{S}_i} \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \leq 1 \iff \\ \iff & \bigwedge_{i=1 \dots n} \min_{t \in \mathcal{S}_i} \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \leq 1 \iff \\ \iff & \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{S}_i} \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \leq 1. \end{aligned}$$

The last result provides a first analytical formulation of the  $\mathbb{M}_n$  feasibility region in the important subcase of Rate Monotonic priorities. It is expressed by the following theorem.

*Theorem 2:* When deadlines are equal to periods, the region of the schedulable task sets  $\mathbb{M}_n$ , as defined by equation (4), is given by:

$$\mathbb{M}_n(T_1, \dots, T_n, D_1, \dots, D_n) |_{D_i=T_i} = \{(C_1, \dots, C_n) \in \mathbf{R}_+^n :$$

$$\bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{S}_i} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\}$$

where  $\mathcal{S}_i = \{rT_j : j = 1 \dots i, r = 1 \dots \lfloor \frac{T_i}{T_j} \rfloor\}$ .

*Proof:* It directly follows from Theorem 1 and equation (4) which defines  $\mathbb{M}_n$ . ■

To understand this result, consider a simple example composed by three tasks. Table I reports the periods  $T_i$  and the sets  $\mathcal{S}_i$ .

$i$	$T_i$	$\mathcal{S}_i$
1	3	{3}
2	8	{3, 6, 8}
3	20	{3, 6, 8, 9, 12, 15, 16, 18, 20}

TABLE I

A 3-TASK SET EXAMPLE OF  $\mathcal{S}_i$  WHEN  $T_1 = 3, T_2 = 8$  AND  $T_3 = 20$ .

The equations we get by expanding Theorem 2 are:

$$\left\{ \begin{array}{ll} C_1 \leq 3 & 3 \in \mathcal{S}_1 \\ C_1 + C_2 \leq 3 & 3 \in \mathcal{S}_2 \\ 2C_1 + C_2 \leq 6 & 6 \in \mathcal{S}_2 \quad \text{plane } \alpha \text{ in Fig. 1} \\ 3C_1 + C_2 \leq 8 & 8 \in \mathcal{S}_2 \quad \text{plane } \beta \text{ in Fig. 1} \\ C_1 + C_2 + C_3 \leq 3 & 3 \in \mathcal{S}_3 \\ 2C_1 + C_2 + C_3 \leq 6 & 6 \in \mathcal{S}_3 \\ 3C_1 + C_2 + C_3 \leq 8 & 8 \in \mathcal{S}_3 \\ 3C_1 + 2C_2 + C_3 \leq 9 & 9 \in \mathcal{S}_3 \\ 4C_1 + 2C_2 + C_3 \leq 12 & 12 \in \mathcal{S}_3 \\ 5C_1 + 2C_2 + C_3 \leq 15 & 15 \in \mathcal{S}_3 \quad \text{plane } \eta \\ 6C_1 + 2C_2 + C_3 \leq 16 & 16 \in \mathcal{S}_3 \quad \text{plane } \theta \\ 6C_1 + 3C_2 + C_3 \leq 18 & 18 \in \mathcal{S}_3 \quad \text{plane } \xi \\ 7C_1 + 3C_2 + C_3 \leq 20 & 20 \in \mathcal{S}_3 \quad \text{plane } \pi \end{array} \right. \quad (5)$$

where the  $\parallel$  symbol denotes the logical OR among the equations in the array, whereas the  $\{$  symbol denotes the logical AND<sup>1</sup>. Fig. 1 shows a graphical representation of the  $\mathbb{M}_3(3, 8, 20)$  region in the  $C$ -space.

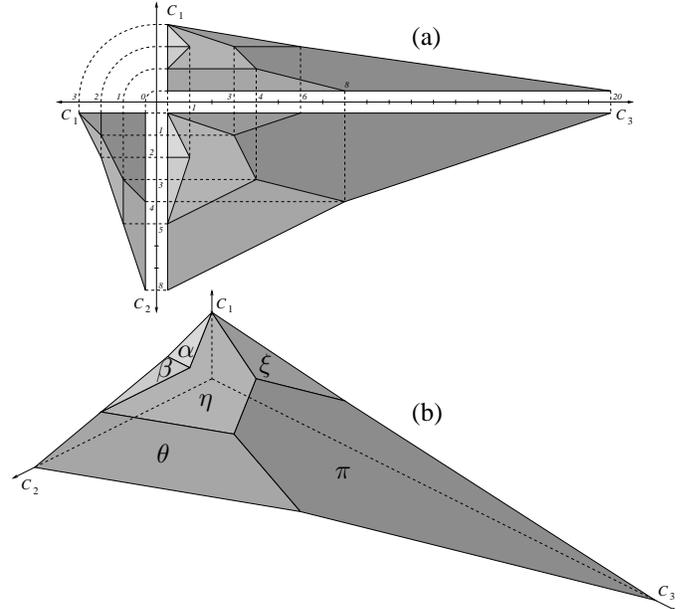


Fig. 1. A view of  $\mathbb{M}_3(3, 8, 20)$  in the  $C$ -space: (a) the projection view, (b) the isometric view.

It is now worth making the following considerations.

- **Observation 1.** The  $\mathbb{M}_n$  region is delimited by planes (hyperplanes in higher dimensions). Every plane equation must be contained in the equation list (5). The opposite is not true; in fact, there can be equations in the list that

<sup>1</sup>The notation ( $\parallel$ ,  $\{$ ) is used instead of the standard ( $\vee$ ,  $\wedge$ ) for a better readability.

are not shown in the picture, because ORed with a more relaxed one.

- **Observation 2.** In Fig. 1 we can distinguish 6 different planes, meaning that in the equation list (5) there are 7 useless equations. Clearly, the situation can change for different values of the periods.
- **Observation 3.** A necessary and sufficient test can be derived by translating the task set  $\Gamma_n = \{(T_1, C_1), \dots, (T_n, C_n)\}$  into a point in the  $C$ -space and then checking whether it belongs to  $\mathbb{M}_n(T_1, \dots, T_n)$ .
- **Observation 4.** Such a formulation is appropriate for making a tunable schedulability test. In fact, every sub-region  $\mathbb{H} \subseteq \mathbb{M}_n$ , obtained by eliminating some of the equations to be OR-ed, is a subset of  $\mathbb{M}_n$ , and hence the feasibility check in  $\mathbb{H}$  will be less complex (i.e., less equations to be checked) and “less necessary” (i.e., smaller region) than the one based on  $\mathbb{M}_n$ .
- **Observation 5.** For large task sets, the number of equations to be checked is huge and is equal to the sum of the number of elements in all  $\mathcal{S}_i$ . When the ratio  $T_n/T_1$  is large, the number of equations is so high that prevents a practical application of Theorem 2 for any guarantee test.
- **Observation 6.** The sum reported in Theorem 2 can also be written as

$$\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t} U_j \leq 1.$$

We note that coefficients  $\left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t}$  are always greater than or equal to 1. Coefficients close to 1 delimit large regions. In particular, if there exists a  $t$  such that  $\left\lceil \frac{t}{T_j} \right\rceil \frac{T_j}{t}$  is equal to 1 for all  $j$ , the  $\mathbb{M}_n$  region becomes  $\sum C_i/T_i \leq 1$ . This is just the mathematical translation of the commonly known behavior of RM when all the periods are in the same harmonic chain [13].

- **Observation 7.** Operation research algorithms could also be applied on the  $\mathbb{M}_n$  region to find the maximum achievable utilization bound. This approach has been followed by Park et al. [15]. However, this method involves a higher number of equations, so it is either slow or not very accurate.

Among the considerations above, the most negative seems to be Observation 5, which says that the high number of equations prevents a practical use of the method. However, as stated in Observation 2, many equations in (5) are useless, so the idea is to reduce the number of equations by eliminating the redundant elements in  $\mathcal{S}_i$ .

Before entering into a detailed discussion of such a reduction process, it is worth noting that such a reduction has been so effective as to make the test not only applicable in practice, but even better than all other tests proposed in the literature. The reduction has been condensed in the next theorem, which is the key contribution of the paper. The theorem and its proof are reported in the next section.

#### IV. FEASIBILITY ANALYSIS IN THE $C$ -SPACE

The following theorem significantly reduces the number of equations that are needed to delimit the  $\mathbb{M}_n$  region.

*Theorem 3:* The region of the schedulable task sets  $\mathbb{M}_n$ , as defined by equation (4), is given by

$$\mathbb{M}_n(T_1, \dots, T_{n-1}, D_1, \dots, D_n) = \{(C_1, \dots, C_n) \in \mathbf{R}_+^n : \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\}$$

where  $\mathcal{P}_i(t)$  is defined by the following recurrent expression:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1} \left( \left\lceil \frac{t}{T_i} \right\rceil T_i \right) \cup \mathcal{P}_{i-1}(t). \end{cases} \quad (6)$$

Before proving the theorem, we first illustrate its application. Then, the formal proof will be given in a dedicated subsection.

The difference between this result and that of Theorem 2 is only the presence of the set  $\mathcal{P}_{i-1}(D_i)$ , instead of  $\mathcal{S}_i$ . This may seem a little change, but it is not. For example, Fig. 2 shows the difference between the sets  $\mathcal{S}_3$  and  $\mathcal{P}_2(D_3)$  for the task set reported in Table I (deadlines are assumed equal to the periods).

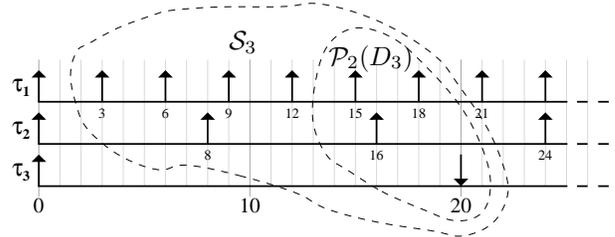


Fig. 2. Comparison between  $\mathcal{S}_3$  and  $\mathcal{P}_2(D_3)$  for the task set shown in Table I.

With the introduction of set  $\mathcal{P}_{i-1}(D_i)$ , the test to check whether a task set belongs to  $\mathbb{M}_n$  becomes not only reasonable, but better than every necessary and sufficient test proposed in the literature. Notice that  $\mathcal{P}_{i-1}(T_i) \subseteq \mathcal{S}_i$  (remember that  $\mathcal{S}_i$  is defined when  $D_i = T_i$ ). This proposition can be formally proved by induction on  $i$  and it is clarified by Fig. 2. This allows to dramatically reduce and bound the time needed for the schedulability test.

Due to the double recurrent form of its definition, the “worst-case” cardinality of a generic  $\mathcal{P}_i(t)$  set is  $2^i$ . We intentionally say “worst-case” cardinality because if the two sets to be joined overlap, the cardinality, of course, reduces.

We note that set  $\mathcal{P}_{i-1}(D_i)$  is only a function of  $D_i$  and of all  $T_j$  for  $j$  from 1 to  $i-1$  and, more in general,  $\mathcal{P}_i(t)$  is a function of time  $t$  and of all the periods  $T_1, \dots, T_i$ . This remark explains why the region in the  $C$ -space is not a function of the lowest priority period  $T_n$ . Fig. 3 shows all the recurrent calls of  $\mathcal{P}_4(D_5)$  in the case of  $T_1 = 9$ ,  $T_2 = 15$ ,  $T_3 = 16$ ,  $T_4 = 36$  and  $D_5 = 100$ . In this figure we can clearly see how the  $\mathcal{P}_j(t)$  definition works. Every set  $\mathcal{P}_j(t)$  is represented by a big grey dot. When  $j \neq 0$ , each set is the union of two sets, and the union relationship is represented by

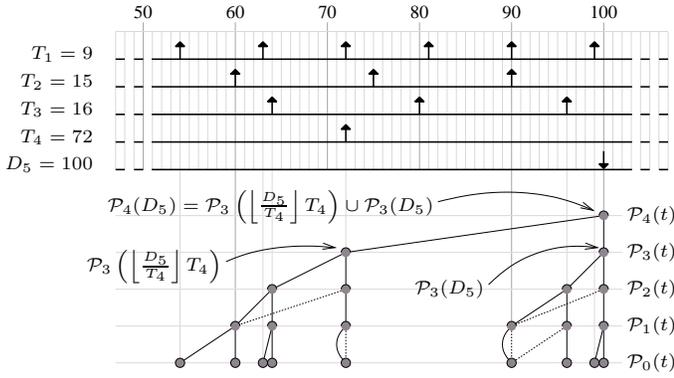


Fig. 3. An example of  $\mathcal{P}_i(t)$ .

a line connecting two sets. A dashed line means that the union does not contribute with new points. Such a case happens, for example, when  $\lfloor \frac{t}{T_j} \rfloor T_j = t$ .

### A. Proof of Theorem 3

To prove Theorem 3 we need the following definitions:

**Definition 1:** A job  $\tau_{ik}$  is said to be **active** at time  $t$  if  $r_{ik} < t < f_{ik}$ .

**Definition 2:** The processor is  **$i$ -busy** at time  $t$  if there exists a job of a task in  $\Gamma_i$  active in  $t$ . More formally, the following function represents the subset of points in  $[0, b]$  where the processor is  $i$ -busy:

$$\text{Busy}(\Gamma_i, b) = \{t \in [0, b] : \exists \tau_{jk} \text{ such that } \tau_{jk} \text{ is active at } t, \tau_j \in \Gamma_i\}.$$

**Definition 3:** The worst-case workload  $W_i(b)$  of the  $i$  highest priority tasks in  $[0, b]$  is the total time the processor is  $i$ -busy in  $[0, b]$ . By extension,  $W_0(b) = 0$  for all  $b$ .

Note that, using the concept of workload, the schedulability condition of  $\tau_i$  can be expressed by

$$C_i + W_{i-1}(D_i) \leq D_i. \quad (7)$$

**Definition 4:** Given the subset  $\Gamma_i$  of the  $i$  highest priority tasks, we define  $\psi_i(b)$  to be the last instant in  $[0, b]$  in which the processor is not  $i$ -busy, that is:

$$\psi_i(b) = \max \{t \in [0, b] \wedge t \notin \text{Busy}(\Gamma_i, b)\}.$$

By Definition 1, the set  $\text{Busy}(\Gamma_i, b)$  is the union of open intervals, hence the set  $[0, b] \setminus \text{Busy}(\Gamma_i, b)$  has always a maximum and so the last idle instant  $\psi_i(b)$  is well defined<sup>2</sup>. This formalism is needed because the point  $\psi_i(b)$  is useful for simplifying the computation of  $W_i(b)$  and to express the FP schedulability condition (7). The following lemma provides a method to easily compute the workload in  $[0, b]$  through the last idle instant  $\psi_i(b)$ .

**Lemma 1:** Given a subset  $\Gamma_i = \{\tau_1, \dots, \tau_i\}$  of the  $i$  highest priority tasks, the workload  $W_i(b)$  can be written as

$$W_i(b) = \sum_{j=1}^i \left\lfloor \frac{\psi_i(b)}{T_j} \right\rfloor C_j + (b - \psi_i(b)).$$

<sup>2</sup>The symbol  $\setminus$  denotes the set difference operator.

*Proof:* From the definition of  $\psi_i(b)$ , we note that no task instance in  $\Gamma_i$  is active at  $\psi_i(b)$  and all the released jobs have been completed at that time. Hence,

$$W_i(\psi_i(b)) = \sum_{j=1}^i \left\lfloor \frac{\psi_i(b)}{T_j} \right\rfloor C_j.$$

Moreover, because the processor is always  $i$ -busy in  $[\psi_i(b), b]$ , the workload of the  $i$  highest priority tasks in such an interval is  $(b - \psi_i(b))$ . Hence, the lemma follows. ■

**Lemma 2:** For any schedulable task subset  $\Gamma_i = \{\tau_1, \dots, \tau_i\}$ ,

$$W_i(b) = \min_{t \in [0, b]} \sum_{j=1}^i \left\lfloor \frac{t}{T_j} \right\rfloor C_j + (b - t). \quad (8)$$

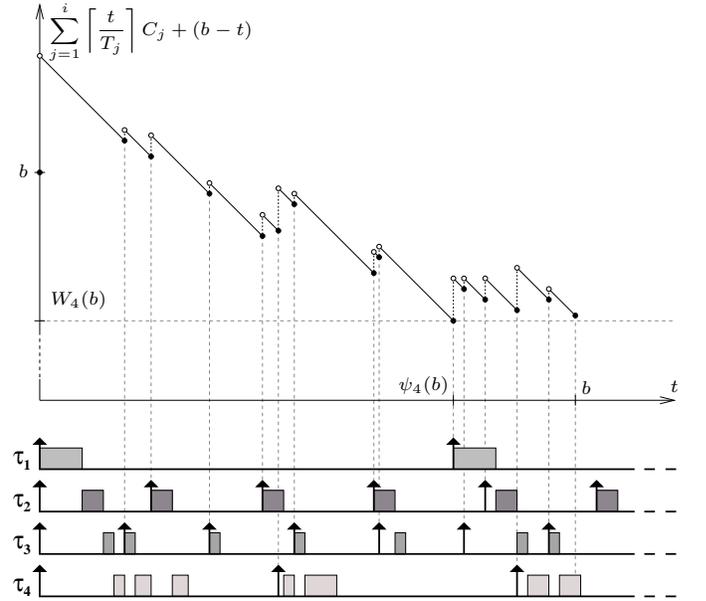


Fig. 4. Lemma 2 interpretation.

*Proof:* We first observe that  $\sum_{j=1}^i \left\lfloor \frac{t}{T_j} \right\rfloor C_j$  is the processor demand in  $[0, t]$ , which is the time required by the tasks to be executed in  $[0, t]$ . So, it must be that

$$\forall t \quad W_i(t) \leq \sum_{j=1}^i \left\lfloor \frac{t}{T_j} \right\rfloor C_j$$

because the processor has no other job to be executed.

Moreover, since in a feasible schedule the workload in  $[t, b]$  (i.e.,  $W_i(b) - W_i(t)$ ) is smaller than the length of the interval, we have that

$$\forall t \in [0, b] \quad W_i(b) - W_i(t) \leq (b - t).$$

or, equivalently,

$$\forall t \in [0, b] \quad W_i(b) \leq \sum_{j=1}^i \left\lfloor \frac{t}{T_j} \right\rfloor C_j + (b - t). \quad (9)$$

Now, since  $\psi_i(b) \in [0, b]$ , for Lemma 1 there exists a value in  $[0, b]$  for which the equality holds, so the lemma follows. ■

The meaning of Lemma 2 is illustrated in Fig. 4, which shows the upper bounding function<sup>3</sup> of the workload for a specific set of four periodic tasks. Notice that the minimum of such a function is  $W_4(b)$  and it falls in  $\psi_4(b)$ .

In other words, the workload  $W_i(b)$  in  $[0, b]$  can be upper estimated by the computation of  $\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t)$  in any  $t \in [0, b]$ . This is not directly useful because this estimation is rough, but the workload can be exactly calculated by the function  $\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t)$  once we know the last idle instant  $\psi_i(b)$ . Unfortunately the complexity moves from the workload estimation to the  $\psi_i(b)$  search. So we now restrict the set of possible values of  $\psi_i(b)$  by the following lemma.

*Lemma 3:* Given a task subset  $\Gamma_i = \{\tau_1, \tau_2, \dots, \tau_i\}$  schedulable by FP, let  $\psi_i(b)$  be the last idle instant in  $[0, b]$  as defined in Definition 4, and let  $\mathcal{P}_i(b)$  be the set of points defined by the following recurrent expression:

$$\begin{cases} \mathcal{P}_0(b) = \{b\} \\ \mathcal{P}_i(b) = \mathcal{P}_{i-1} \left( \left\lfloor \frac{b}{T_i} \right\rfloor T_i \right) \cup \mathcal{P}_{i-1}(b). \end{cases}$$

Then,

$$\psi_i(b) \in \mathcal{P}_i(b).$$

*Proof:*

We demonstrate the lemma by induction on  $i$ .

**Initial Step.** If  $i = 1$ , we have to prove that, for a schedulable task  $\tau_1$ ,  $\psi_1(b) \in \mathcal{P}_1(b)$  for all  $b$ . We observe that, in this case,

$$\mathcal{P}_1(b) = \mathcal{P}_0 \left( \left\lfloor \frac{b}{T_1} \right\rfloor T_1 \right) \cup \mathcal{P}_0(b) = \left\{ \left\lfloor \frac{b}{T_1} \right\rfloor T_1, b \right\}.$$

Since  $\tau_1$  is schedulable, then the last idle instant  $\psi_1(b)$  can only be:

- 1) at  $\lfloor b/T_1 \rfloor T_1$ , if the last instance in  $[0, b]$  of  $\tau_1$  is active at  $b$ ;
- 2) at  $b$ , otherwise.

Both values are in  $\mathcal{P}_1(b)$  and the initial step is proved.

**Inductive Step.** If  $\psi_i(b) \in \mathcal{P}_i(b)$  for all  $b$ , we have to prove that, given a schedulable task subset  $\Gamma_{i+1} = \{\tau_1, \dots, \tau_{i+1}\}$ , then  $\psi_{i+1}(b) \in \mathcal{P}_{i+1}(b)$  for all  $b$ .

Let us consider the time interval  $[\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$ . In this interval two things can happen:

- 1) the processor is  $(i+1)$ -busy in the whole interval;
- 2) there exists an instant of time at which the processor is not  $(i+1)$ -busy.

In the first case,  $\psi_{i+1}(b) = \psi_{i+1}(\lfloor b/T_{i+1} \rfloor T_{i+1})$  because in  $[\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$  the processor always runs a task in  $\Gamma_{i+1}$ . Moreover, it must be  $\psi_{i+1}(\lfloor b/T_{i+1} \rfloor T_{i+1}) = \psi_i(\lfloor b/T_{i+1} \rfloor T_{i+1})$  otherwise the last instance of  $\tau_{i+1}$  in  $[0, \lfloor b/T_{i+1} \rfloor T_{i+1}]$  would miss its deadline at  $\lfloor b/T_{i+1} \rfloor T_{i+1}$ , contradicting the hypothesis of  $\tau_{i+1}$  schedulability.

In the second case, let  $x \in [\lfloor b/T_{i+1} \rfloor T_{i+1}, b]$  be an instant of time where no tasks in  $\Gamma_{i+1}$  are active. Since at time  $x$  the  $\lfloor b/T_{i+1} + 1 \rfloor^{\text{th}}$  job of  $\tau_{i+1}$  is terminated,  $\tau_{i+1}$  is never active in  $[x, b]$ . This implies that  $\psi_{i+1}(b) = \psi_i(b)$ .

<sup>3</sup>Black dots indicate the value of the function in the discontinuities.

Merging the two cases we get:

$$\psi_{i+1}(b) = \psi_i \left( \left\lfloor \frac{b}{T_{i+1}} \right\rfloor T_{i+1} \right) \vee \psi_{i+1}(b) = \psi_i(b).$$

For the inductive hypothesis, we have that:  $\psi_i(\lfloor b/T_{i+1} \rfloor T_{i+1}) \in \mathcal{P}_i(\lfloor b/T_{i+1} \rfloor T_{i+1})$  and  $\psi_i(b) \in \mathcal{P}_i(b)$ . Hence,

$$\psi_{i+1}(b) \in \mathcal{P}_i \left( \left\lfloor \frac{b}{T_{i+1}} \right\rfloor T_{i+1} \right) \cup \mathcal{P}_i(b)$$

and finally, for the  $\mathcal{P}_{i+1}(b)$  definition:

$$\psi_{i+1}(b) \in \mathcal{P}_{i+1}(b)$$

which proves the inductive step and the lemma.  $\blacksquare$

*Lemma 4:* Given a task subset  $\Gamma_i = \{\tau_1, \dots, \tau_i\}$  schedulable by FP and the set  $\mathcal{P}_i(b)$  as defined in equation (6), the workload  $W_i(b)$  is

$$W_i(b) = \min_{t \in \mathcal{P}_i(b)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b-t).$$

*Proof:* By observing that  $\mathcal{P}_i(b) \subseteq [0, b]$ , the lemma directly follows from Lemmæ 1, 2, and 3.  $\blacksquare$

We are now ready to prove the theorem.

**Proof of Theorem 3.** We have to prove the equivalence between the following two sentences:

- 1) for all  $i = 1 \dots n$ ,  $\tau_i$  is schedulable by a fixed priority algorithm;

$$2) \bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$

Without loss of generality, we can assume that, for the schedulability of task  $\tau_i$ , all tasks  $\tau_1, \dots, \tau_{i-1}$  are schedulable. In this case, the schedulability condition of the single task  $\tau_i$  can be written as:

$$C_i + W_{i-1}(D_i) \leq D_i$$

where  $W_{i-1}(D_i)$  is the workload of the first  $i-1$  tasks in the interval  $[0, D_i]$ . Using Lemma 4, such a condition of individual schedulability can be written as:

$$C_i + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \leq D_i$$

$$C_i + \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j - t \leq 0$$

$$\bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$

Hence, the schedulability condition for all the tasks is clearly given by

$$\bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

as required by Theorem 3.

In the next paragraphs two examples are illustrated to extend the applicability of this result: first one considers a non-RM priority assignment and the second one deals with deadlines smaller than periods.

### B. Example with non-RM priorities

Let us consider the same tasks shown in Table I. However, in this example, we invert the priorities of the tasks with period 8 and 20 (we still assume  $D_i = T_i$ ). In particular The resulting tasks set is ordered as follows:

$i$	$T_i$	$D_i$	$\mathcal{P}_{i-1}(D_i)$
1	3	3	{3}
2	20	20	{18, 20}
3	8	8	{0, 6, 8}

By expanding the equations given by Theorem 3, which provides a necessary and sufficient condition, we get the following inequalities, which describe the exact schedulability region:

$$\left\{ \begin{array}{l} C_1 \leq 3 \\ \parallel \\ 6C_1 + C_2 \leq 18 \\ 7C_1 + C_2 \leq 20 \\ \parallel \\ C_3 \leq 0 \\ 2C_1 + C_2 + C_3 \leq 6 \\ 3C_1 + C_2 + C_3 \leq 8 \end{array} \right.$$

Notice that the inequality  $C_3 \leq 0$  would force  $C_3$  to be zero. Such a strict constraint is not a problem because it is ORED with more relaxed ones.

As was done for the Rate Monotonic case, we can now draw the schedulability region in the  $C$ -space, formerly defined as  $\mathbb{M}_n$ . The result is depicted in Fig. 5.

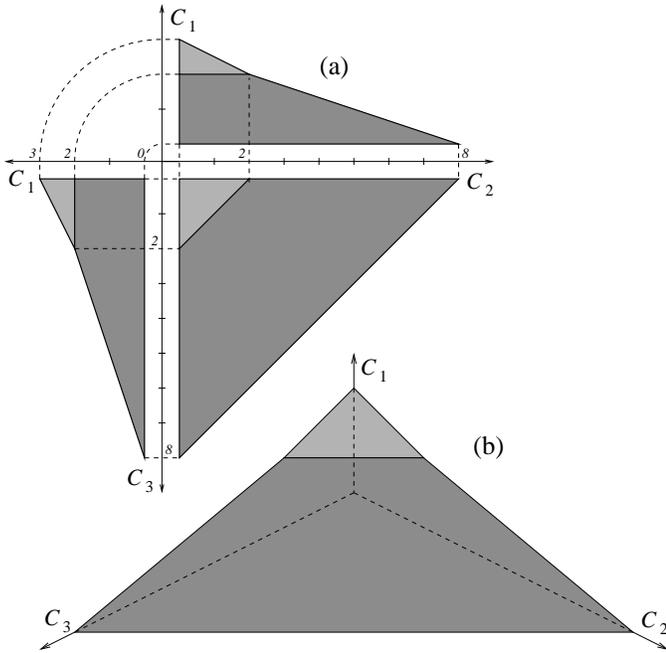


Fig. 5. The schedulability region when  $FP \neq RM$ : (a) the projection view and (b) the isometric view.

It is worth noticing that this region is smaller than the RM one (see Fig. 1). This is a clear consequence of the RM optimality: a task set schedulable by a generic fixed priority assignment is also schedulable by Rate Monotonic.

### C. Example with deadlines smaller than periods

We now consider a task set with deadlines smaller than periods. Tasks are the same as those in Table I, but deadlines  $D_2$  and  $D_3$  are decreased by one unit of time, leading to the following task set:

$i$	$T_i$	$D_i$	$\mathcal{P}_{i-1}(D_i)$
1	3	3	{3}
2	8	7	{6, 7}
3	20	19	{15, 16, 18, 19}

Applying Theorem 3 we get the following set of inequalities:

$$\left\{ \begin{array}{l} C_1 \leq 3 \\ \parallel \\ 2C_1 + C_2 \leq 6 \\ 3C_1 + C_2 \leq 7 \\ \parallel \\ 5C_1 + 2C_2 + C_3 \leq 15 \\ 6C_1 + 2C_2 + C_3 \leq 16 \\ 6C_1 + 3C_2 + C_3 \leq 18 \\ 7C_1 + 3C_2 + C_3 \leq 19 \end{array} \right.$$

The resulting schedulability region  $\mathbb{M}_n$  is shown in Fig. 6. We notice that the region achieved in this case is smaller than that shown in Fig. 1, confirming that shortening deadlines reduces schedulability.

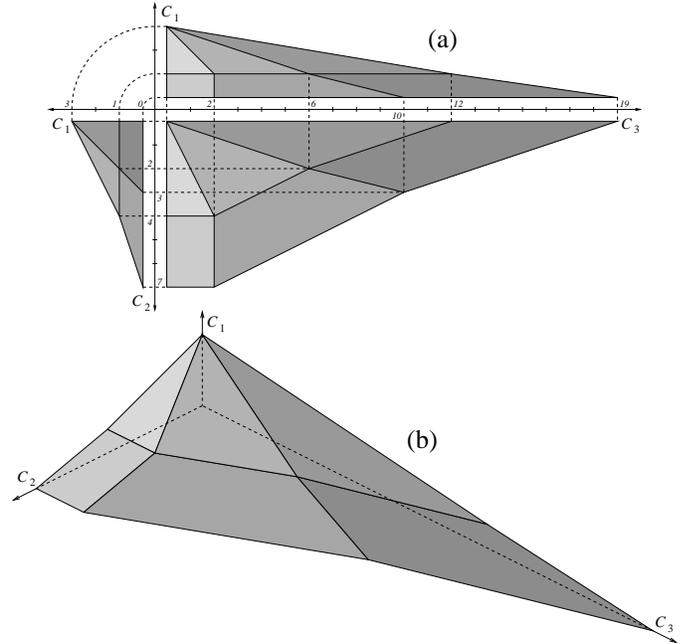


Fig. 6. The schedulability region when  $D_i < T_i$ : (a) the projection view and (b) the isometric view.

## V. THE HYPERPLANES $\delta$ -EXACT TEST

In this section we first show how the necessary and sufficient test is derived and then we describe the method to make it tunable.

As we saw in the last section, the FP schedulability condition in Theorem 3 can be equivalently expressed as

$$\forall i = 1 \dots n \quad C_i + W_{i-1}(D_i) \leq D_i$$

where the workload  $W_{i-1}(D_i)$  is given by Lemma 4:

$$W_{i-1}(D_i) = \min_{t \in \mathcal{P}_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t)$$

and  $W_0(D_1) = 0$  by extension.

Using this formulation, the pseudo-C code of the necessary and sufficient test can be written as follows:

```

boolean FPTest( $\Gamma_n$ ) {
int  $i$ ;

for ( $i=1; i \leq n; i++$ )
  if ( $C_i + \text{WorkLoad}(i-1, D_i) > D_i$ )
    return false;
return true;
}

```

Now we focus our attention on the function  $\text{WorkLoad}(i, b)$ , which is the workload of the  $i$  highest priority tasks in  $[0, b]$ , also called  $W_i(b)$  previously. From Lemma 4 we know that

$$W_i(b) = \min_{t \in \mathcal{P}_i(b)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t)$$

and then, from the definition of  $\mathcal{P}_i(b)$ , we can write (in the following expressions we define  $f = \lfloor b/T_i \rfloor$  and  $c = \lceil b/T_i \rceil$ ):

$$W_i(b) = \min \left\{ \begin{array}{l} \min_{t \in \mathcal{P}_{i-1}(fT_i)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t), \\ \min_{t \in \mathcal{P}_{i-1}(b)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t), \end{array} \right\} \quad (10)$$

where we split the set  $\mathcal{P}_i(b)$  in the two subsets which compose it.

We now write these expressions in a more meaningful form. By noting that the  $i^{\text{th}}$  element in the first sum is always equal to  $fC_i$  (due to the schedulability of  $\tau_i$ ) the first element of equation (10) can be written as

$$\begin{aligned} \min_{t \in \mathcal{P}_{i-1}(fT_i)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) &= \\ &= fC_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= b + fC_i - fT_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (fT_i - t) = \\ &= b - f(T_i - C_i) + W_{i-1}(fT_i) \end{aligned}$$

from the result of Lemma 4. Similarly, the second element in equation (10) can be written as:

$$\begin{aligned} \min_{t \in \mathcal{P}_{i-1}(b)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) &= \\ &= cC_i + \min \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t) = \\ &= cC_i + W_{i-1}(b) \end{aligned}$$

Hence, equation (10) can be expressed in a recurrent form as follows:

$$W_i(b) = \min\{b - f(T_i - C_i) + W_{i-1}(fT_i), cC_i + W_{i-1}(b)\}. \quad (11)$$

Such a recurrent expression of  $W_i(b)$  directly follows from the recurrent definition of  $\mathcal{P}_i(b)$ . As in the  $\mathcal{P}_i(b)$  definition the two sets  $\mathcal{P}_{i-1}(\lfloor b/T_i \rfloor)$  and  $\mathcal{P}_{i-1}(b)$  could overlap (see Fig. 3), it can happen that, for particular values of  $j$  and  $t$ , two calls of  $W_j(t)$  could return the same value. In this case, following both branches would be a waste of time. This can be avoided by keeping track of the execution flow through a global variable which can be used to prune all the useless branches. The resulting algorithm is shown below:

```

double last $\psi$ [BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

double WorkLoad(int  $i$ , double  $b$ ) {
  int  $f, c$ ;
  double branch0, branch1;

  if ( $i \leq 0$ ) return 0;
  if ( $b \leq \text{last}\psi[i]$ ) /* if WorkLoad( $i, b$ ) already computed */
    return lastWorkLoad[ $i$ ]; /* don't go further */
   $f = \lfloor b/T_i \rfloor$ ;  $c = \lceil b/T_i \rceil$ ;
  branch0 =  $b - f(T_i - C_i) + \text{WorkLoad}(i-1, fT_i)$ ;
  branch1 =  $cC_i + \text{WorkLoad}(i-1, b)$ ;
  last $\psi$ [ $i$ ] =  $b$ ;
  lastWorkLoad[ $i$ ] = min(branch0, branch1);
  return lastWorkLoad[ $i$ ];
}

```

We note that the  $i^{\text{th}}$  element of the array lastWorkLoad keeps track of the call  $\text{WorkLoad}(i, \text{last}\psi[i])$ . The necessary and sufficient algorithm obtained in this way, called the Hyperplanes Exact Test (HET), is quite more efficient than the response time based algorithm. The performance comparison is presented in Section VI.

#### A. The $\delta$ -HET test

The tunable test  $\delta$ -HET is obtained by reducing the  $\mathcal{P}_i(b)$  set as a function of an additional parameter  $\delta$ , as follows:

$$\mathcal{P}_i(b, \delta) = \begin{cases} \{b\} & \forall \delta \\ \mathcal{P}_{i-1}\left(\left\lfloor \frac{b}{T_i} \right\rfloor T_i, \delta\right) \cup \mathcal{P}_{i-1}(b, \delta) & \text{if } b\delta \geq T_i \\ \mathcal{P}_{i-1}\left(\left\lfloor \frac{b}{T_i} \right\rfloor T_i, \delta\right) & \text{otherwise} \end{cases}$$

By this definition it is easy to prove the following properties of the  $\mathcal{P}_i(b, \delta)$  set:

$$\begin{aligned} \delta_1 \leq \delta_2 &\iff \mathcal{P}_i(b, \delta_1) \subseteq \mathcal{P}_i(b, \delta_2) \\ \mathcal{P}_i(b, 1) &= \mathcal{P}_i(b). \end{aligned}$$

Similar properties also hold for the workload expressed by Lemma 4. If we define

$$W_i^{(\delta)}(b) = \min_{t \in \mathcal{P}_i(b, \delta)} \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j + (b - t)$$

then we have that

$$\begin{aligned} \delta_1 \leq \delta_2 &\iff W_i^{(\delta_1)}(b) \geq W_i^{(\delta_2)}(b) \\ W_i^{(1)}(b) &= W_i(b). \end{aligned}$$

Both properties follow directly from the definition of  $W_i^{(\delta)}$ . In particular, the first property derives from the observation that the minimum cannot decrease when computed on a smaller set.

The  $\delta$ -HET test can then be derived by substituting  $\mathcal{P}_i(b)$  with  $\mathcal{P}_i(b, \delta)$ . The resulting algorithm for computing the new workload function is illustrated below.

```

double last $\psi$ [BIG_ENOUGH];
double lastWorkLoad[BIG_ENOUGH];

double WorkLoad(int  $i$ , double  $b$ , double  $\delta$ ) {
  int  $f$ ,  $c$ ;
  double branch0, branch1;

  if ( $i \leq 0$ )
    return 0;
  if ( $b \leq$  last $\psi$ [ $i$ ]) /* if WorkLoad( $i$ ,  $b$ ) already computed */
    return lastWorkLoad[ $i$ ]; /* don't go further */
   $f = \lfloor b/T_i \rfloor$ ;  $c = \lceil b/T_i \rceil$ ;
  branch0 =  $b - f(T_i - C_i) +$  WorkLoad( $i - 1$ ,  $f T_i$ );
  if ( $T_i \leq b \delta$ )
    branch1 =  $c C_i +$  WorkLoad( $i - 1$ ,  $b$ );
  else
    branch1 = branch0; /* one branch is cut! */
  last $\psi$ [ $i$ ] =  $b$ ;
  lastWorkLoad[ $i$ ] = min(branch0, branch1);
  return lastWorkLoad[ $i$ ];
}

```

## VI. PERFORMANCE OF $\delta$ -HET

In this section, we compare the  $\delta$ -HET test with the common schedulability tests proposed in the literature for the RM algorithm, when deadlines are equal to periods. The performance of a test is evaluated in terms of both acceptance ratio and complexity. The acceptance ratio is measured by the number of accepted task sets with respect to those accepted by a necessary and sufficient test. In particular, if  $U$  denotes the set of task sets for which there exists a feasible schedule, let  $S^* \subseteq U$  be those task sets which are schedulable with fixed priorities (and hence RM schedulable), and let  $S^T \subseteq S^*$  be those task sets which are guaranteed by a sufficiency test  $T$ . Then the acceptance ratio  $\text{acceptanceRatio}(T)$  of a test  $T$  is computed as follows:

$$\text{acceptanceRatio}(T) = \frac{|S^T|}{|S^*|}$$

where the operator  $|\cdot|$  indicates “the number of elements”. From this definition it follows that  $\text{acceptanceRatio}(T) \leq 1$  for all  $T$ , and  $\text{acceptanceRatio}(T) = 1$  for the response-time analysis and all the necessary and sufficient tests.

The complexity of a test is measured by counting the number of innermost loop iterations. Formally, we define  $\text{stepsNumber}(T, \Gamma)$  as the number of innermost loop steps required to compute the guarantee test  $T$  on the tasks set  $\Gamma$ . Moreover, we model the computation time of a guarantee test  $T$  as a random variable  $\text{steps}(T)$ , defined by the following cumulative distribution function (c.d.f.):

$$F_{\text{steps}(T)}(x) = P\{\text{stepsNumber}(T, \Gamma) \leq x\} \quad \Gamma \in U$$

where  $P\{\cdot\}$  denotes the probability of an event. This model is useful because the maximum and the average number of

iteration steps can easily be extracted from the probability density function (p.d.f.), which is

$$f_{\text{steps}(T)} = \frac{dF_{\text{steps}(T)}}{dx}.$$

In our experiments, simulations have been performed by generating  $10^8$  task sets, each composed by 8 tasks. Task periods  $T_i$  were randomly extracted in  $[1, 1000000]$  (with uniform distribution) and computation times  $C_i$  were computed as random variables in  $[0, T_i]$  (also with uniform distribution). Fig. 7 illustrates the results of a first experiment, which compares three necessary and sufficient tests: the classical Response Time Analysis (RTA) of Audsley et al. [4], the Response Time analysis Improved (RTI) by Sjödin et al. [9], and the Hyperplanes Exact Test (HET) presented in this paper. The figure plots the probability density achieved for the three tests.

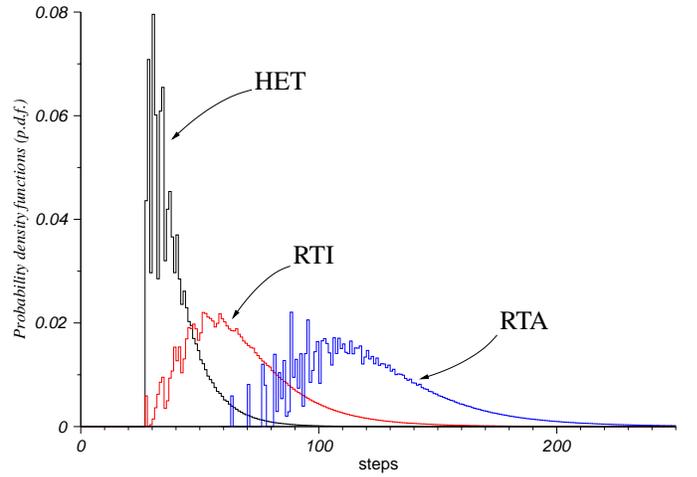


Fig. 7. p.d.f. of  $\text{steps}(RTA)$ ,  $\text{steps}(RTI)$  and  $\text{steps}(HET)$ .

We notice that the HET test, although pseudo-polynomial, is significantly faster than the others, not only for the average case, but also, and especially, for its worst case. The appreciable “noise” of the probability density is not due to the low number of sets ( $10^8$ ), but it comes from the intrinsic structure of the algorithm.

Fig. 8 shows the result of another experiment, in which we evaluated the dependency of the average number of steps on the number of tasks in the set. As we can clearly see, although for all tests the average number of steps increases exponentially, in the case of the HET algorithm the speed of growth is significantly smaller.

In a third experiment, we evaluated the performance of the  $\delta$ -HET test with respect to several other tests proposed in the literature, such as the Response Time Analysis (RTA) [4], the Response Time analysis Improved (RTI) [9], the Liu and Layland test (LL) [1], and the Hyperbolic Bound (HB) [12]. The  $\delta$ -HET has been executed for several values of  $\delta \in [0.5, 1]$ . The comparison is made in terms of both acceptance ratio and complexity, on a universe of  $10^6$  tasks sets consisting of 8 tasks.

In particular, each test is characterized by two values (that is, the average number of steps and the acceptance ratio),

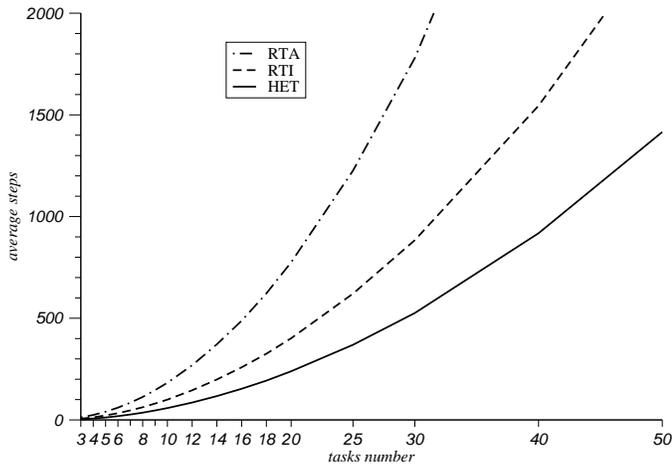


Fig. 8. Average number of steps as a function of the number of tasks.

thus it is represented as a point in a plane, having  $\text{steps}(T)$  and  $\text{acceptanceRatio}(T)$  as coordinates. In such a plane, the best area for a guarantee test is the one located around the upper-left corner, where the acceptance ratio is high and the

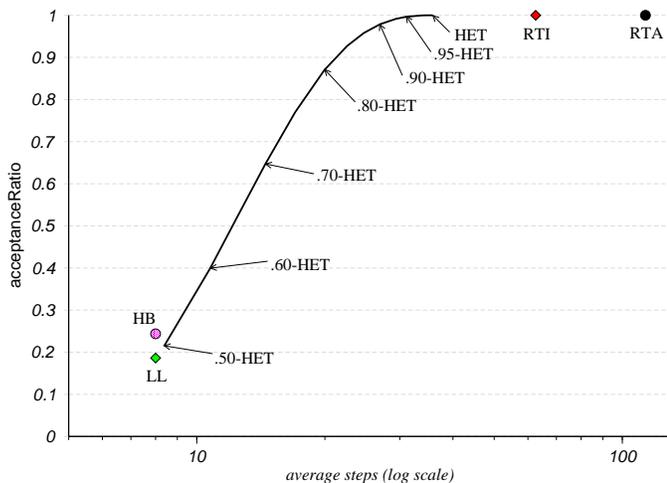


Fig. 9. Comparison of guarantee tests.

As we can see, the performance of the  $\delta$ -HET test covers all the intermediate positions in the plane. In particular when  $n = 8$ , for  $\delta = 0.5$  the  $\delta$ -HET test has a performance similar to the one of the LL-test, whereas for  $\delta = 1$  (exact analysis) it is still significantly better than the RTI test.

## VII. EXTENSIONS

This section extends the proposed analysis to the case of shared resources and non-preemptive scheduling. We then show some additional application of the proposed methodology.

### A. Shared resources

The blocking times due to resource sharing can be taken into account by introducing, for each task  $\tau_i$ , a blocking term

$B_i$  to be added to the task computation time [7]. Hence, the schedulability condition (7) can be extended as follows:

$$C_i + B_i + W_{i-1}(D_i) \leq D_i. \quad (12)$$

Using the same reasoning, Theorem 3 can also be extended as follows, to take blocking times into account.

**Theorem 4:** A task set  $\Gamma$  composed by tasks  $\tau_i = (C_i, T_i, D_i, B_i)$  is schedulable **if**:

$$\bigwedge_{i=1 \dots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t - B_i. \quad (13)$$

Clearly, this condition is more stringent than the one expressed by Theorem 3. To better understand the impact of this result, we consider an example with the same task set as before, in which tasks  $\tau_1$  and  $\tau_2$  have blocking times  $B_1 = B_2 = 1$ .

$i$	$T_i$	$D_i$	$B_i$	$\mathcal{P}_{i-1}(D_i)$
1	3	3	1	{3}
2	8	7	1	{6, 7}
3	20	19	0	{15, 16, 18, 19}

Applying Theorem 4, the schedulability region in the  $C$ -space can be described by the following inequalities:

$$\left\{ \begin{array}{l} C_1 \leq 2 \\ \parallel \\ 2C_1 + C_2 \leq 5 \\ 3C_1 + C_2 \leq 6 \\ \parallel \\ 5C_1 + 2C_2 + C_3 \leq 15 \\ 6C_1 + 2C_2 + C_3 \leq 16 \\ 6C_1 + 3C_2 + C_3 \leq 18 \\ 7C_1 + 3C_2 + C_3 \leq 19 \end{array} \right.$$

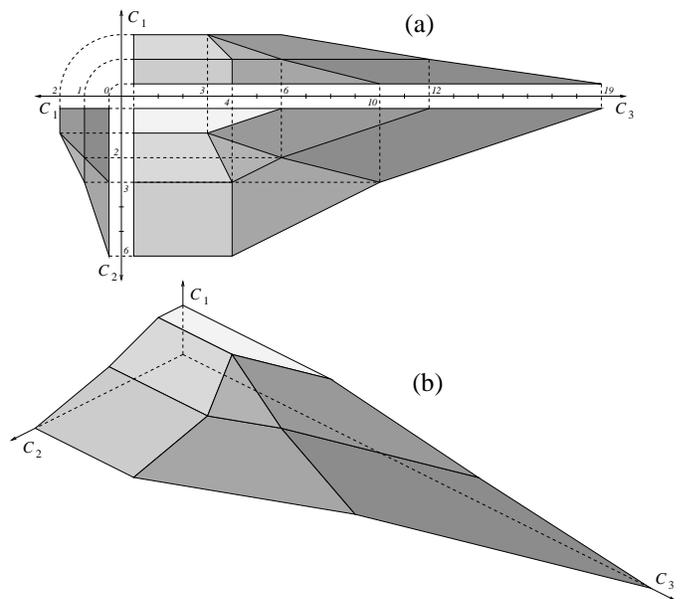


Fig. 10. The schedulability region in presence of shared resources: (a) the projection view and (b) the isometric view.

The shape of the region for the given example is illustrated in Fig. 10. Again, we notice that this region is smaller than that shown in Fig. 6, achieved in the absence of blocking times.

As a last remark, notice that the last theorem provides only a sufficient condition, because when a blocking time is present, the worst-case scenario can be different than the synchronous (classical Liu and Layland) one.

### B. Non-preemptive scheduling

The proposed analysis can be immediately extended to non-preemptive scheduling by observing that non-preemptive scheduling is just a special case of resource sharing, where all tasks share a single resource for their entire execution. Hence, Theorem 4 can be applied in this case if blocking times are computed as

$$B_i = \max_{j>i} C_j \quad (14)$$

We now show how the schedulability region of task set in Table I is reduced in the non-preemptive case. By applying Theorem 4 with equations 14, we get:

$$\left\{ \begin{array}{l} C_1 \leq 3 - \max\{C_2, C_3\} \\ \parallel \\ 2C_1 + C_2 \leq 6 - C_3 \\ 3C_1 + C_2 \leq 8 - C_3 \\ \parallel \\ 5C_1 + 2C_2 + C_3 \leq 15 \\ 6C_1 + 2C_2 + C_3 \leq 16 \\ 6C_1 + 3C_2 + C_3 \leq 18 \\ 7C_1 + 3C_2 + C_3 \leq 20 \end{array} \right.$$

and then:

$$\left\{ \begin{array}{l} C_1 + C_2 \leq 3 \\ C_1 + C_3 \leq 3 \\ \parallel \\ 2C_1 + C_2 + C_3 \leq 6 \\ 3C_1 + C_2 + C_3 \leq 8 \\ \parallel \\ 5C_1 + 2C_2 + C_3 \leq 15 \\ 6C_1 + 2C_2 + C_3 \leq 16 \\ 6C_1 + 3C_2 + C_3 \leq 18 \\ 7C_1 + 3C_2 + C_3 \leq 20 \end{array} \right.$$

Simplyfing the redundant equations, we finally get:

$$\left\{ \begin{array}{l} C_1 + C_2 \leq 3 \\ C_1 + C_3 \leq 3 \end{array} \right. \quad (15)$$

Equations 15 summarize some known properties of the non-preemptive scheduling algorithms. For example, it is well known that every single computation time  $C_i$  must be not greater than the smallest period. In the considered example, the smallest period ( $T_1$ ) is 3 and the found condition deny all the  $C_i$  to be greater than 3.

## VIII. CONCLUSIONS AND FUTURE WORK

In this paper we presented a novel approach for analyzing the schedulability of periodic task sets under the Rate Monotonic priority assignment. Such an approach allowed us to precisely describe the feasibility region in the space of task computation times (the  $C$ -space) and to derive a tunable guarantee test ( $\delta$ -HET), by which we can balance acceptance ratio and complexity. Such a tunability property of the  $\delta$ -HET test is important in those cases in which the performance of a polynomial time test is not sufficient for achieving high

processor utilization, and the overhead introduced by exact tests is too high for an on-line admission control.

We believe that the proposed formulation opens a novel direction in the schedulability analysis of fixed priority systems, allowing further research in this domain. As a future work, we plan to investigate the case where task computation times are considered as random variables with known probability distribution. In this case, the probability to meet the deadlines of a task set can be computed by the integral of the  $C_i$  probability density on the  $\mathbb{M}_n$  region.

Another interesting situation that can be addressed by the proposed method deals with the case in which not all the computation times are fixed, but we have some freedom to select the size of some tasks (for instance when using imprecise computation models). In this condition, the HET approach can be used to decide the best values of those free variables in order to improve schedulability. In fact, fixing a  $C_i$  value is equivalent to cutting the  $\mathbb{M}_n$  region in the  $C$ -space at a certain coordinate (this can easily be visualized in Fig. 1).

## REFERENCES

- [1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 40–61, 1973.
- [2] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate-monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proceedings of the 10<sup>th</sup> IEEE Real-Time Systems Symposium*, 1989, pp. 166–172.
- [3] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal, British Computer Society*, vol. 29, no. 5, pp. 390–395, October 1986.
- [4] N. C. Audsley, A. Burns, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," *Software Engineering Journal*, December 1993.
- [5] Y. Manabe and S. Aoyagi, "A feasibility decision algorithm for rate monotonic scheduling of periodic real-time tasks," in *Proceedings of the 1<sup>st</sup> Real-Time Technology and Applications Symposium*, May 1995, pp. 297–303.
- [6] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.
- [7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.
- [8] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," in *IEEE Proceedings of the Euromicro Workshop on Real-Time Systems*, June 1996, pp. 29–33.
- [9] M. Sjödin and H. Hansson, "Improved response-time analysis calculations," in *Proceedings of the 19<sup>th</sup> IEEE Real-Time Systems Symposium*, December 1998, pp. 399–409.
- [10] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Systems*, vol. 9, pp. 207–239, 1995.
- [11] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.
- [12] E. Bini, G. C. Buttazzo, and G. M. Buttazzo, "Rate monotonic scheduling: The hyperbolic bound," *IEEE Transactions on Computers*, vol. 52, no. 7, pp. 933–942, July 2003.
- [13] T.-W. Kuo and A. K. Mok, "Load adjustment in adaptive real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1991.
- [14] D. Chen, A. K. Mok, and T.-W. Kuo, "Utilization bound re-visited," in *Proceedings of the 6<sup>th</sup> Real-Time Computing Systems and Applications*, 1999, pp. 295–302.
- [15] D. Park, S. Natarajan, and M. J. Kim, "A generalized utilization bound test for fixed-priority real-time scheduling," in *Proceedings of the 2<sup>nd</sup> International Workshop on Real-Time Systems and Applications*, October 1995, pp. 73–76.
- [16] H. Liu and X. Hu, "Efficient performance estimation for general real-time task systems," in *IEEE/ACM International Conference on Computer Aided Design*, 2001, pp. 464–471.

PLACE  
PHOTO  
HERE

**Enrico Bini** is a PhD student in Computer Engineering at Scuola Superiore Sant'Anna in Pisa (Italy). In 2000, he received the Laurea degree in Computer Engineering from the 'Università di Pisa'. In the same year he received the Licenza degree from the Scuola Superiore Sant'Anna.

In 1999 he was a visiting student at Technische Universiteit Delft, in the Netherlands. In 2001 he was with Ericsson Lab Italy in Roma. In 2003 he was a visiting student at University of North Carolina at Chapel Hill, collaborating with prof. Sanjoy Baruah.

His interests cover scheduling algorithms, real-time operating systems, embedded systems design and optimization algorithms.

PLACE  
PHOTO  
HERE

**Giorgio C. Buttazzo** is an Associate Professor of Computer Engineering at the University of Pavia, Italy. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, in Philadelphia. From 1991 to 1998, he held a position of Assistant Professor at the

Scuola Superiore S. Anna of Pisa, doing research on robot control systems and real-time scheduling.

His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He is a member of the IEEE and the IEEE Computer Society.